Catching up and moving ahead

- Toward understanding Java Interfaces
  - Language idiom, standard idea
  - Code to an interface, not an implementation (meaning?)
  - Understand trade-offs in implementations

- Using mathematical notation in an intuitive way
  - Formalisms for another course, intuition helpful
  - No hand-waving, but building a foundation

- Preview of linked-structures and trees
  - For after midterm exam

What is a java.util.List in Java?

- Collection of elements, operations?
  - Add, remove, traverse, ...
  - What can a list do to itself?
  - What can we do to a list?

- Why more than one kind of list: Array and Linked?
  - Useful in different applications
  - How do we analyze differences?
  - How do we use them in code?

What’s the Difference Here?

- How does find-a-track work? Fast forward?

Analyze Data Structures

```java
public double removeFirst(List<String> list) {
    double start = System.currentTimeMillis();
    while (list.size() != 1) {
        list.remove(0);
    }
    double end = System.currentTimeMillis();
    return (end-start)/1000.0;
}
List<String> linked = new LinkedList<String>();
List<String> array = new ArrayList<String>();
double ltime = splicer.removeFirst(splicer.create(linked,100000));
double atime = splicer.removeFirst(splicer.create(array,100000));

- Time taken to remove the first element?
  - Who gets off a line/queue first?
```
Removing first element

<table>
<thead>
<tr>
<th>Size</th>
<th>Link</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.003</td>
<td>0.045</td>
</tr>
<tr>
<td>20</td>
<td>0.001</td>
<td>0.173</td>
</tr>
<tr>
<td>30</td>
<td>0.001</td>
<td>0.383</td>
</tr>
<tr>
<td>40</td>
<td>0.003</td>
<td>0.680</td>
</tr>
<tr>
<td>50</td>
<td>0.002</td>
<td>1.074</td>
</tr>
<tr>
<td>60</td>
<td>0.002</td>
<td>1.510</td>
</tr>
<tr>
<td>70</td>
<td>0.003</td>
<td>2.071</td>
</tr>
<tr>
<td>80</td>
<td>0.003</td>
<td>2.704</td>
</tr>
<tr>
<td>90</td>
<td>0.004</td>
<td>3.449</td>
</tr>
<tr>
<td>100</td>
<td>0.007</td>
<td>4.220</td>
</tr>
</tbody>
</table>

Interfaces

- What is an interface? What does Google say?
  - Term overloaded even in English
  - What is a Java Interface?
- Abstraction that defines a contract/construct
  - Implementing requires that certain methods exist
    - For example, Comparable interface?
  - Programming to the interface is enabling
    - What does Collections.sort actually sort?
- IDE helps be putting in stubs as needed
  - Let Eclipse be your friend

Middle Index Removal

```java
public double removeMiddleIndex(List<String> list) {
    double start = System.currentTimeMillis();
    while (list.size() != 1) {
        list.remove(list.size() / 2);
    }
    double end = System.currentTimeMillis();
    return (end - start) / 1000.0;
}
```

- What operations could be expensive here?
  - Explicit: size, remove
  - Implicit: find n-th element

Remove middle elt/index

<table>
<thead>
<tr>
<th>Size</th>
<th>Link</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.105</td>
<td>0.323</td>
</tr>
<tr>
<td>20</td>
<td>0.417</td>
<td>0.759</td>
</tr>
<tr>
<td>30</td>
<td>0.684</td>
<td>1.162</td>
</tr>
<tr>
<td>40</td>
<td>1.841</td>
<td>2.383</td>
</tr>
<tr>
<td>50</td>
<td>3.226</td>
<td>5.339</td>
</tr>
<tr>
<td>60</td>
<td>5.086</td>
<td>7.674</td>
</tr>
<tr>
<td>70</td>
<td>8.079</td>
<td>1.938</td>
</tr>
<tr>
<td>80</td>
<td>11.365</td>
<td>3.862</td>
</tr>
</tbody>
</table>
ArrayList and LinkedList as ADTs

- As an ADT (abstract data type) ArrayList supports
  - Constant-time or O(1) access to the k-th element
  - Amortized linear or O(n) storage/time with add
    - Total storage used in n-element vector is approx. 2n, spread over all accesses/additions (why?)
  - Add/remove in middle is "expensive" O(n), why?

- LinkedList as ADT
  - Constant-time or O(1) insertion/deletion anywhere, but...
  - Linear or O(n) time to find where, sequential search

- Linked good for add/remove at front
  - Also for splicing into middle, also for 'sparse' structures

Inheritance and Interfaces

- Interfaces provide method names and parameters
  - The method signature we can expect and thus use!
  - What can we do to an ArrayList? To a LinkedList?
  - What can we do to a Map or Set or PriorityQueue?
  - java.util.Collection is an interface

- Abstract classes can have core, duplicated code
  - If we can add one object to a [set,map,list], can we add an entire list of objects? java.util.AbstractCollection
  - If iterate can we remove? Convert to array? Obtain size?
    - Where do we read about this?

Nancy Leveson: Software Safety

- Founded the field
- Mathematical and engineering aspects
  - Air traffic control
  - Microsoft word
    - "C++ is not state-of-the-art, it's only state-of-the-practice, which in recent years has been going backwards"
- Software and steam engines once deadly dangerous?

- THERAC 25: Radiation machine killed many people

Big-Oh, Θ-notation: concepts & caveats

- Count how many times "simple" statements execute
  - In the body of a loop, what matters? (e.g., another loop?)
  - Assume simple statements take a second, cost a penny, ...
    - What’s good, what’s bad about this assumption?
    - Alternatives?

- If a loop is inside a loop:
  - Tricky because the inner loop can depend on the outer, use math and reasoning

- In real life: cache behavior, memory behavior, swapping behavior, library gotchas, things we don’t understand,...
More on O-notation, big-Oh

- Big-Oh hides/obscures some empirical analysis, but is good for general description of algorithm
  - Allows us to compare algorithms in the limit
    - 20N hours vs N^2 microseconds: which is better?
- O-notation is an upper-bound, this means that N is O(N), but it is also O(N^2); we try to provide tight bounds. Formally:
  - A function g(N) is O(f(N)) if there exist constants c and n such that g(N) < cf(N) for all N > n.

Notations for measuring complexity

- O-notation/big-Oh: O(n^2) is used in algorithmic analysis, e.g., CompSci 130 at Duke. Upper bound in the limit
  - Correct to say that linear algorithm is O(n^2), but useful?
- Omega is lower bound: \( \Omega(n \log n) \) is a lower bound for comparison based sorts
  - Can’t do better than that, very hard to prove
- Sedgewick/Wayne uses tilde notation \( \sim n^2 \) means leading term is \( n \) squared
  - We’ll use this, but call it big-Oh in notational abuse

Simple examples of loop counting

```java
for(int k=0; k < list.size(); k += 1) {
    list.set(k, list.get(k)+1);
}
//-----
for(int k=0; k < list.size(); k += 1)
    for(int j=k+1; j < list.size(); j += 1)
        if (list.get(j).equals(list.get(k)))
            matches += 1;
//---
for(int k=0; k < list.size(); k += 1)
    for(int j=k; j < list.size(); j *= 2)
        value += 1;
```

Multiplying and adding big-Oh

- Suppose we do a linear search then do another one
  - What is the complexity? O(n) + O(n)
  - If we do 100 linear searches? 100*O(n)
  - If we do n searches on an array of size n? n * O(n)
- Binary search followed by linear search?
  - What are big-Oh complexities? Sum?
  - What about 50 binary searches? What about n searches?
- What is the number of elements in the list (1,2,2,3,3,3); (1,2,2,3,3,3,4,4,4,4,4)?
  - What about (1,2,2, ..., n,n,...,n)?
What is big-Oh about?

- Intuition: avoid details when they don’t matter, and they don’t matter when input size (N) is big enough
  - For polynomials, use only leading term, ignore coefficients
    \[ y = 3x \quad y = 6x - 2 \quad y = 15x + 44 \]
    \[ y = x^2 \quad y = x^2 - 6x + 9 \quad y = 3x^2 + 4x \]
  - The first family is \( O(n) \), the second is \( O(n^2) \)
- Intuition: family of curves, generally the same shape
- More formally: \( O(f(n)) \) is an upper-bound, when \( n \) is large enough the expression \( cf(n) \) is larger
- Intuition: linear function: double input, double time, quadratic function: double input, quadruple the time

Some helpful mathematics

- \[ 1 + 2 + 3 + 4 + \ldots + N \]
  \[ N(N+1)/2 \text{, exactly } = N^2/2 + N/2 \text{ which is } O(N^2) \text{ why?} \]
- \[ N + N + N + \ldots + N \text{ (total of N times)} \]
  \[ N^2 \text{ which is } O(N^2) \]
- \[ N + N + N + \ldots + N + \ldots + N \text{ (total of 3N times)} \]
  \[ 3N^2 \text{ which is } O(N^2) \]
- \[ 1 + 2 + 4 + \ldots + 2^N \]
  \[ 2^{N+1} - 1 = 2 \times 2^N - 1 \text{ which is } O(2^N) \]
- Adding \( 2^{N+1} \) elements to an ArrayList?
  \[ 1 + 2 + \ldots + 2^N + 2^{N+1} = 2^{N+2} - 1 \]
  \[ 4x2^N - 1 \text{ which is } O(2^N) \]
  resizing + copy = total (let \( x = 2^N \))

Helpful formulae

- We always mean base 2 unless otherwise stated
  - What is \( \log(1024) \)?
  - \( \log(xy) = \log(x) + \log(y) \)
  - \( \log(x^y) = y \log(x) \)
  - \( \log(2^{x}) = x \log(2) = n \)
  - \( 2^{(\log x)} = n \)
- Sums (also, use sigma notation when possible)
  - \[ 1 + 2 + 4 + 8 + \ldots + 2^i = 2^{i+1} - 1 = \sum_{i=0}^{k} 2^i \]
  - \[ 1 + 2 + 3 + \ldots + n = n(n+1)/2 = \sum_{i=1}^{n} i \]
  - \[ a + ar + ar^2 + \ldots + ar^{n-1} = a(r^n - 1)/(r-1) = \sum_{i=0}^{n-1} ar^i \]

Fran Allen

- IBM Fellow, Turing Award
  - Optimizing compilers
- Taught high school for two years, then Master’s degree and IBM
  - Teachers excited me to learn
  
  I’ve always felt that theory without practice is maybe nice and maybe pretty, but it’s not going to influence computing as much as the practice side. But the practice has to be backed up with the ability to talk about it, reason about it, and formulate it so that it can be reproduced.